# Partial Visibility Polygon with Semi-Transparent Objects

Mostafa Nouri Baygi[*] Mohammad Ghodsi[†]

**Abstract**

In this paper, we study partial visibility (or p-visibility) of a scene containing semi-transparent (convex) obstacles through which the light can pass partly. We define the p-visibility polygon and give algorithms to compute it for any query point. Then, we use the same technique to update the p-visibility polygon of a moving point, and to compute the maximum number of intersected objects by a ray emanating from a query point.

## 1 Introduction

We study the concept of partial visibility (or *p-visibility*) and present an algorithm to compute p-visibility polygon for a query point. Roughly speaking, the p-visibility polygon is the set of visible regions around a viewer when there are some semi-transparent objects in the scene.

P-Visibility can be applicable in many areas. For example, as Fulek *et al.* [2] mentioned, according to a model of wireless positioning service patented by Liu and Hung [3], the signal sent by a sensor can penetrate only at most a certain number, $k$, of obstacles and will not be received by the base station if there are more than $k$ obstacles between the sensor and the base station.

Fulek *et al.* [2] studied a problem related to p-visibility. For a set $S$ of $n$ objects in the plane, and a point $p$, they defined $\tau(p, S)$ as the maximum number of objects that are intersected by all the rays emanated from $p$. Likewise, they defined $\tau(S)$ as the minimum value of $\tau(p, S)$ over all points $p$. Their problem is to provide an upper and lower bound for the value of $\tau(n)$, which is the maximum of $\tau(S)$ over all sets $S$ of $n$ objects. They also showed how $\tau(S)$ can be computed in $O(n^4 \log n)$.

In addition to algorithms for computing and updating the p-visibility polygon, we show how to compute $\tau(q)$ for a query point $q$. Briefly, we give the following results:

1. In the presence of some semi-transparent objects with total complexity of $n$, we compute any desired p-visibility polygon of a query point, in $O(n^2 \log(\sqrt{m}/n)/\sqrt{m} + |PVP(q)|)$ query time, using $O(m)$ space. Here $|PVP(q)|$ is the total size of the p-visibility polygons.
2. For a moving point, we maintain the p-visibility polygon of the point, in $O(n^2 \log(\sqrt{m}/n)/\sqrt{m})$ time for each change, and detect the first place that a change occurred in the same time.
3. For a query point $q$, we compute $\tau(q)$ in $O(n^2 \log(\sqrt{m}/n)/\sqrt{m})$.

In the above formulae, $n^2 \le m \le n^4$

## 2 Preliminaries

Assume that a set $S$ includes $l$ disjoint semi-transparent convex polygons, called objects, in the plane with total complexity of $n$. We need to compute the visible portion of the plane from an observer point. This problem is similar to computing the visibility polygon of a point.

Since the objects are semi-transparent, the light can partially pass through them, i.e., its intensity degrades when the light passes through an object. Assume that the light intensity only decreases when it enters an object. This way, we have different areas in the plane, each is visible with a different intensity. The problem is to compute these regions.

With the notion of p-visibility, for a point $q$ and a parameter $k$, we define $k$-$PVP(q)$ as the set of points in the plane whose connecting line segments to $q$ intersects at most $k$ objects. Obviously, $k$-$PVP(q)$ contains $j$-$PVP(q)$ for $j < k$. For $k \ge \tau(S)$, $k$-$PVP(q)$ consists of all the plane and $0$-$PVP(q)$ is the well-known visibility polygon of $q$. The main problem in this paper is to compute $k$-$PVP(q)$.

Let $r_q$ be a ray emanating from $q$ and let $\tau(r_q)$ be the set of objects intersected by $r_q$. For each $q$, we define $\tau(q) = \max_{r_q} \tau(r_q)$. This is the same as $\tau(q, S)$ proposed by Fulek *et al.* [2]. We can define $\tau(q)$ in another way: $\tau(q)$ is the smallest $k$, such that $k$-$PVP(q)$ is all the plane.

## 3 P-visibility polygon computation

In this section, we present an algorithm that computes $k$-$PVP(q)$. In this problem, we have a set $S$ of convex polygons, with total complexity of $O(n)$, called objects, and a query point $q$. We can preprocess $S$ so

[*]Department of Computer Engineering, Sharif University of Technology, nourybay@ce.sharif.edu

[†]Department of Computer Engineering, Sharif University of Technology, ghodsi@sharif.edu

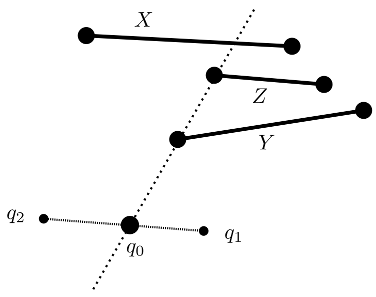

Figure 1: A tangent shows the place where the p-visibility changes occurs.

that for a given $q$ $k$-$PVP(q)$ can be efficiently computed for any desired value of $k$.

We first describe the algorithm that uses $O(n^4)$ space, and then extend it to the general case with the space–query-time tradeoff.

### 3.1 Logarithmic query time

In order to compute $k$-$PVP(q)$ for a query point, we partition the plane into regions that all the points in each region have similar $k$-$PVP(q)$. The similarity is defined by the order of visibility of visible edges of the objects in $S$ from observer. We denote this ordered list of visible edges by $k$-$\mathcal{PVP}(q)$. The following lemma helps identify the desired partition.

**Lemma 1** *For a point $q$ in the plane, $k$-PVP$(q)$ for any $k$ changes only if $q$ moves across the tangent line of a pair of edges.*

**Proof.** Let $q_1$, $q_2$ be two points in the same cell in the arrangement, $\mathcal{A}$, of tangent lines of all pairs of the edges of the objects and $k$-$\mathcal{PVP}(q_1)$, $k$-$\mathcal{PVP}(q_2)$ differ, for some fixed value of $k$. Consider moving point $q$ from $q_1$ towards $q_2$. When $q$ moves, $k$-$\mathcal{PVP}(q)$ changes until it equals to $k$-$\mathcal{PVP}(q_2)$. Let $q_0$ be the first point at which $k$-$\mathcal{PVP}(q)$ changes. This change is in the form of removing a visible edge from or adding a new previously invisible edge to the list. Assume that it is in the latter form, so the object $Z$ is added between $X$, $Y$. This means that before we reach $q_0$, the number of objects between $q$ and any point on $C$ was at least $k+1$, but in $q_0$, the number of objects between $q$ to a point on $C$ is at most $k$. This event happens only when a segment previously blocked $C$ and then, in $q_0$, it becomes visible. As it can be seen in Figure 1, $q_0$ is on the tangent line of two segments ($B$, $C$ in this example), which contradicts the assumption that $q_1$, $q_2$ are in the same cell. Similarly, the other case leads to a contradiction. $\square$

As above lemma shows, we need to construct the arrangement $\mathcal{A}$ and compute $k$-$\mathcal{PVP}(p)$ for a point $p$ in each cell. By this approach we can prove the following theorem.

**Theorem 2** *Given a point $q$, we can compute $k$-PVP$(q)$ for $0 \leq k \leq \tau(q,S)$, in $O(\log n + |k$-PVP$(q)|$ query time, while using $O(\tau(S)n^4)$ space and $O(\tau(S)n^4 \log n)$ preprocessing time.*

**Proof.** We construct the arrangement $\mathcal{A}$ and obtain a tour visiting all the cells of $\mathcal{A}$, such that each edge of $\mathcal{A}$ is visited at most twice, by a depth-first traversal of the cells of $\mathcal{A}$. Then, we select an arbitrary cell and compute $k$-$\mathcal{PVP}$ for all $0 \leq k \leq \tau(S)$ for an arbitrary point in that cell and store them, in a set of persistent red-black tree [5], each element of the set for one value of $k$. Since $k$-$\mathcal{PVP}$ is an ordered list of objects, it can be inserted in a binary search tree without any ambiguity. Afterwards, we move to the next cell in the tour and compute new lists of $k$-$\mathcal{PVP}$ in that cell. $k$-$\mathcal{PVP}$ in adjacent cells of $\mathcal{A}$ differ in 1 position, so we can store the new $k$-$\mathcal{PVP}$'s in the persistent data structures easily in $O(\tau(S)\log n)$.

In the query time, for a point $q$, we identify the cell in $\mathcal{A}$ that $q$ lies in, and search in the persistent red-black tree for related $k$-$\mathcal{PVP}$. We can report this data structure as the ordered list of visible objects, or compute $k$-$PVP(q)$ precisely in the order of size of $k$-$PVP(q)$.

The construction of $\mathcal{A}$, which consists of $O(n^2)$ lines, takes $O(n^4)$ and in the same time we can create a tour. Computing $k$-$\mathcal{PVP}$ for $0 \leq k \leq \tau(S)$ in the first cell and storing them in the data structure takes $O(\tau(S)n\log n)$ time. Computing $k$-$\mathcal{PVP}$ for other cells in $\mathcal{A}$ each takes $O(\tau(S)\log n)$ and totally $O(\tau(S)n^4 \log n)$. We should preprocess $\mathcal{A}$ for a point location data structure [1] which can be done in $O(n^4 \log n)$. In the query time, a point location and a search in the persistent data structure is required to find the stored $k$-$\mathcal{PVP}(q)$, all of which takes $O(\log n)$. We can construct the actual $k$-$PVP(q)$ based on $k$-$\mathcal{PVP}(q)$. $\square$

In above theorem, we assume that $k$ is a parameter which is specified at query time, so we compute all the different p-visibility polygons and stored them in the persistent data structure. But if $k$ is determined in the preprocessing step, we can reduce the memory space and preprocessing time considerably. In this case, we only compute p-visibility polygon for that specified $k$. This way the memory space (resp. preprocessing time) is reduced to $O(n^4)$ (resp. to $O(n^4 \log n)$).

In the above result, we can compute $k$-$\mathcal{PVP}(q)$ for several values of $k$, without any change in the query time (except for reporting). This is because the search in the persistent data structure should be done only once and the associated lists can be returned easily.

Here, we provide two notes about how to optimize the arrangement $\mathcal{A}$. First, when we draw the line through $a_1b_1$ as a tangent for segments $A = (a_1, a_2)$ and $B = (b_1, b_2)$, the portion between $a_1$ and $b_1$ can be removed without any problem. This is because when

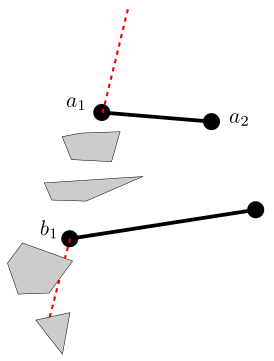

Figure 2: Optimization of the arrangement by reducing tangent lines. In this example, $k_{max} = 3$

$q$ crosses from this part of the tangent, no changes occurs in the p-visibility polygons. Second, if the maximum value of $k$ is determined in the preprocessing step, $k_{max}$, and we want to draw the tangent line through $a_1 b_1$ as before, we can continue the tangent from $b_1$ as long as the points on it see at most $k$ objects before $a_1$. The same is true for the other portion of the tangent (see Figure 2).

## 3.2 Space–query-time tradeoff

In this section, we show how to modify the previous method, and reduce the preprocessing space in the expense of an increase in the query time. The following lemma which is a modified version of the cutting theorem, in [4] is the main tool that helps achieve this.

**Lemma 3** *Given a set of $n$ lines in the plane, we can partition the plane into $O(r^2)$ triangles, for $1 \le r \le n$, such that each triangle is intersected by $O(n/r)$ lines and any arbitrary new line intersects at most $O(r)$ triangles.*

**Theorem 4** *For any query point $q$, we can compute $k$-PVP$(q)$ for all $0 \le k \le \tau(S)$, in $O(n^2 \log(\sqrt{m}/n)/\sqrt{m})$ query time, while using $O(\tau(S)m)$ space and $O(\tau(S)m \log(\sqrt{m}/n))$ preprocessing time, for an arbitrary $n^2 \le m \le n^4$.*

**Proof.** The set, $P$, of the vertices of the objects consists of $O(n)$ points and in the dual plane, this corresponds to a set of $O(n)$ lines, denoted by $P^*$. We start by constructing a cutting of size $O(r^2)$ for this set, such that each triangle of the cutting intersects $O(n/r)$ lines of $P^*$.

For a query point $q$ in the primal plane, its dual line $q^*$ intersects $O(r)$ triangles, and the intersection of $q^*$ with these triangles, in the primal plane, partitions the plane into $O(r)$ co-centered disjoint double wedges, totally covering all the plane. Therefore, it is enough to compute the $k$-$PVP(q)$ in each double

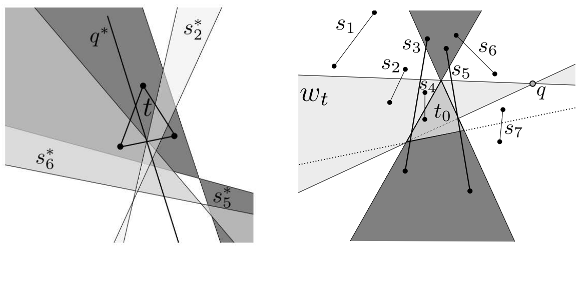

Figure 3: (a) In the dual plane, the triangle $t$, dual of $t_0$, is intersected by $q^*$. The dual of some edges of objects are also shown. (b) The arrangement of $t_0$, $q$ and edges of objects in the primal plane.

wedge, or equivalently, in each triangle intersected by $q^*$.

In Figure 3(a), the triangle $t$ in the dual plane is shown which is intersected by $q^*$. The triangle $t$ is also intersected by $O(n/r)$ lines which are dual to the end-points of $O(n/r)$ edges of objects in $S$. We call these segments the *closed segments* of $t$, denoted by the set $CS_t$. At least the dual of one end-point of each segment in $CS_t$ intersects $t$. There are also some segments in $O$ which are not in $CS_t$, but may appear in $k$-$\mathcal{PVP}(q)$, for example $s_8$ in Figure 3. We call these segments the *open segments* of $t$, denoted by the set $OS_t$. $OS_t$ consists of all the segments whose dual double wedge contains $t$ completely.

The triangle $t_0$, whose dual is $t$, as in Figure 3(b), partitions the plane into three regions. A region, which is brighter in the figure, is the region that $q$ and at least one end-point of each segment of $CS_t$ lie (for example segments $s_6$ and $s_7$). In contrast, we have two regions, which is shaded in the figure, and contains at most one end-point of each segment of $CS_t$ and both end-points of the other segments of $O$. If an end-point of a segment $o_i$ lies in a shaded region and the other end-point in the other shaded region, $o_i$ belongs to $OS_t$ (for example segments $s_3$ and $s_5$).

It is not hard to see that the segments in $OS_t$ partition the bright region, the region which contains $q$, into $|OS_t|+1$ subregions. Let $R_t = \{r_t^1, \ldots, r_t^{|OS_t|+1}\}$ denote the set of subregions. In each subregion $r_t^k$, a set of segments $CS_t^k \subset CS_t$ is contained.

Since the view around $q$ is bounded to $w_t$, we can imagine each segment in $OS_t$ as an infinite line. Therefore, to identify the cells with uniform $k$-$\mathcal{PVP}(q)$, we only construct the arrangement of tangents for each pair of segments in $CS_t$ and not $OS_t$. We only consider the objects of segments in $OS_t$ as obstacles.

With this approach, we can use the previous method with logarithmic time in each double wedge. For each triangle $t$, we spend $O(\tau(S)(n/r)^4)$ space and $O(\tau(S)(n/r)^4 \log(n/r))$ preprocessing time.

To complete the argument, we should say how to compute $CS_t$ and $OS_t$. $CS_t$ of complexity $O(n/r)$, which is already computed during the construction of the cutting $\mathcal{R}$, is the set of segments with an end-point whose dual intersects $t$.

It is also notable that, we cannot compute $OS_t$ for each triangle independently, since the size of $OS_t$ may be large, compared to $O((n/r)^4)$, which we can spend for each triangle. The elements of $OS_t$ for each $t$, is very similar to $OS_{t'}$ of an adjacent triangle $t'$. The differences are in the elements of $CS_{t'}$ and $CS_t$, that is some segments in $CS_t$ may be removed from $OS_{t'}$ and some segments in $CS_{t'}$ may be added to $OS_{t'}$ to produce $OS_t$.

At query time, we find the cell in $t^*$ that contains $q$ by a point location and return the associated $k$-$\mathcal{PVP}$ in $O(\log(n/r))$ time for each triangle $t$ that is intersected by $q^*$. We can easily compute $k$-$\mathcal{PVP}(q)$ bounded by $w_t$ in the same time. Summing up these amounts for $O(r)$ triangles, we can compute $k$-$\mathcal{PVP}(q)$ in $O(r\log(n/r))$ time.

In summary, the total preprocessing time and space are $O(\tau(S)n^4\log(n/r)/r^2)$ and $O(\tau(S)n^4/r^2)$ respectively and the query time is $O(r\log(n/r))$. If the used space is denoted by $m$ we can prove the claim. □

## 4 Applications

In this section, we apply the techniques used in the previous section to two related problems and give solutions for them.

### 4.1 Maximum intersecting objects

**Theorem 5** *For any query point $q$ we can compute $\tau(q)$ in $O(n^2\log(\sqrt{m}/n)/\sqrt{m})$, using $O(m)$ space and $O(O(m\log(\sqrt{m}/n)))$ preprocessing time, for $n^2 \le m \le n^4$*

**Proof.** Consider the cutting we used before in the dual plane for the vertices of the objects. For each point $q$, the intersection of $q'$ and triangle $t$, in the primal plane corresponds to the double wedge $w_t$. Here we should change the data structures, so that instead of storing the actual $k$-$\mathcal{PVP}(q)$ in each cell of the arrangements, we only store the maximum value of objects, that are intersected by any ray emanated from any point inside $w_t$, denoted by $\tau_t(q)$. To compute $\tau(q)$, we should compute $\tau_t(q)$, for all triangles $t$ that are intersected by $q'$, and choose the maximum value among them, which can be done in the same query time as before. □

### 4.2 P-visibility polygon of a moving point

**Theorem 6** *For a moving point $q$ in the plane, which moves on a straight line, we can detect the first place where $k$-$\mathcal{PVP}(q)$ changes for some $k$ and update $k$-$\mathcal{PVP}(q)$ in $O(\frac{n^2}{\sqrt{m}}(\log\frac{\sqrt{m}}{n}))$. The preprocessing time and space are respectively $O(\tau(S)m\log(\sqrt{m}/n))$ and $O(\tau(S)m)$.*

**Proof.** Assume we can use $O(\tau(S)n^4)$ space for computing $k$-$PVP(q)$. In this case, $q$ is a point which lies in a cell $c$ in the arrangement of tangent lines. $c$ is a convex polygon, therefore for a straight line, we can identify in $O(\log n)$, from which edge of $c$, $q$ leaves it. Once $q$ leaves $c$, $k$-$PVP(q)$ may change for some value of $k$. We can easily detect this event and update $k$-$\mathcal{PVP}(q)$ based on the edge $q$ crosses.

For the case that we use tradeoff, consider line $q^*$ in the dual plane. When $q$ moves on a straight line, $q^*$ rotates around a fixed center. In each triangle, we can easily detect the first change in the visibility similar to the previous case that was described. Here we need to choose the first place from these $O(r)$ places that changes occurred. All of these, can be accomplished in $O(r\log(n/r))$. Substituting $r$ with $n^2/\sqrt{m}$ proves the theorem. □

## 5 Conclusion

In this paper, we introduced the p-visibility concept and presented an algorithm that computes p-visibility polygon of a query point in logarithmic time. We then extended the algorithm to reduce the space usage, but in the expense of an increase in the query time.

Finally, we used the method to solve two related problems: updating p-visibility polygon of a moving point and computing the maximum number of objects that are intersected by a ray emanated from a query point. For the future works, we intend to solve this problem: for a set of objects $S$, find a point $p$ such that $\tau(S) = \tau(p)$ in optimal time.

## References

[1] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geometry*, 9:145–158, 1993.

[2] R. Fulek, A. F. Holmsen, and J. Pach. Intersecting convex sets by rays. *Discrete Comput. Geometry*, 42(3):343–358, 2009.

[3] C.-T. Liu and T.-Y. Hung. Method of building a locating service for a wireless network environment. patent no. 7203504, April 2007. www.freepatentsonline.com/7203504.html.

[4] M. Nouri and M. Ghodsi. Space–query-time tradeoff for computing the visibility polygon. In *FAW '09*, pages 120–131. Springer-Verlag, 2009.

[5] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986.